Санкт-Петербургский Государственный Университет

АВДЕЕВ Иван Владимирович

Выпускная квалификационная работа

Разработка семантического языка запросов для анализа больших данных

Уровень образования: бакалавриат

Направление *02.03.02 «Фундаментальная информатика и информационные технологии»*

Основная образовательная программа: *СВ.5003.2016 «Программирование и информационные технологии»*

Профиль: «Автоматизация научных исследований»

Научный руководитель:

Доктор физико-математических наук, профессор

кафедры компьютерного моделирования

и многопроцессорных систем

Андрианов Сергей Николаевич

Рецензент:

Кандидат физико-математических наук, доцент

кафедры теории систем управления

электрофизической аппаратурой

Едаменко Николай Семенович

Санкт-Петербург

2020

Содержание

Введение 5

YT и YQL 7

Постановка задачи 10

Обзор литературы 12

Глава 1. Подготовка к решению задачи 14

1.1 Анализ имеющихся данных 14

1.2 Стек технологий 16

1.2.1 Язык запросов 16

1.2.2 Интерфейс 18

1.2.3 YQL и YT 19

Глава 2. Обзор существующих решений 20

2.1 Встроенные возможности языка JavaScript 20

2.2 libxml by Google 22

2.3 Выводы 22

Глава 3. Реализация 24

3.1 Чтение XPath запросов 24

3.2 Поиск данных в дереве 27

3.2.1 Идея алгоритма 27

3.2.2 Реализация 28

3.3 Проблемы, появившиеся при реализации поиска 29

3.3.1 Неограниченное количество узлов между двумя определенными 29

3.3.2 Индексация узлов в дереве 30

3.4 Реализация интерфейса 32

3.4.1 Отправка JSON дерева на страницу с решением поиска по XPath 32

3.4.2 Отображение подходящих запросу узлов 33

3.4.3 Хранение узлов и быстрый доступ к ним 33
3.5 Поиск в YQL 35
Выводы 37
Что уже сделано? 37
Интерфейс для отображения 38
Работа с несколькими документами одновременно 39
Поиск в YQL 40
Дальнейшие улучшения приложения 42
Оптимизация запросов 42
Оптимизация алгоритма 43
Проблема множественных XPath запросов 43
Заключение 44
Список литературы 45
1. Литература 45
2. Ссылки на документации 45
3. Ссылки на сайты с полезной информацией 49
Приложение 50
Приложение А. Список терминов, употребляемых в работе 50
Приложение В. Примеры кода 51
В.1. Пример: работа библиотеки xpath-analyzer 51
В.2. Код функции поиска в документе узлов, подходящих запросу 53
В.3. Код функции поиска, рассматривающий случай неограниченного количества узлов между двумя данными 53
В.4. Код функции, сохраняющий данные по узлам в хэш-таблицы для повторного поиска 54

Приложение С. Примеры рабочего приложения 55
С.1. Начальный интерфейс при загрузке страницы 55
С.2. Поиск в документе 55
С.3. Отображение нескольких документов на странице 56
С.4. Поиск в нескольких документах сразу 56

Введение

Поиск данных в большом наборе - интересная и в то же время сложная задача. Чтобы решить ее, необходимо определить:

- как и по какому критерию проводить поиск;
- оптимальный алгоритм, реализующий поиск;
- что нужно считать выходным результатом.

Каждый день компания Яндекс (далее просто Компания) обрабатывает миллиарды поисковых запросов и выдает на них какой-то ответ. Чтобы оставаться в топе (быть лучшей среди всех поисковых систем), Компания использует множество технологий для улучшения поиска (например, алгоритм Multi-Armed Bandits - алгоритм выдачи поиска, который «подмешивал» в топ поиска к самым популярным сайтам сайты-новички).

Для того, чтобы все эти технологии работали, необходимо каким-то образом сохранять информацию о поведении пользователя на странице выдачи поиска - необходимо сохранять клики, скроллы, сохранять счётчики и т.д. Это помогает понять, где больше всего происходит активности от пользователей на той или иной странице выдачи поиска. У каждой кнопки/ссылки на странице есть счетчик: он показывает, сколько раз пользователи перешли по какой-либо ссылке или нажали на определенную кнопку.

На первых порах эти данные хранились в большой строковой переменной, и для того, чтобы достать из нее какие-то определенные данные, разработчикам нужно было всего лишь написать простой скрипт, который находит подстроку в строке. Это не занимало много времени и сильно не отвлекало разработчиков на написание лишнего кода.

Со временем данные становились все сложнее и больше: они приобрели структуру сложного и большого документа. Для каждой страницы создается

документ, который хранит в себе все данные о ссылках на странице, на которые можно перейти (ссылки на счетчики, ссылки на метрики и т.д.).

На данный момент, все данные, которые содержат в себе информацию о счетчиках и метриках, хранятся в платформе YT.

YT - основная платформа для хранения и обработки больших объемов данных в компании Яндекс [27]. Это MapReduce система, направленная на широкий спектр вычислительных задач в области Big Data. Более подробную информацию можно найти в источнике [27] и в видеозаписи конференции «Yet another Conference» в докладе «YT — новая платформа распределённых вычислений».

MapReduce — модель распределенных вычислений, представленная компанией Google, используемая для параллельных вычислений над очень большими, вплоть до нескольких петабайт, наборами данных в компьютерных кластерах (источник - Википедия).

С увеличением сложности и размера данных о счётчиках увеличилось и время их обработки, поиска нужных данных. Если раньше достаточно было написать простой скрипт, который находил подстроку в строке, то теперь необходимо реализовывать обход документа в поисках необходимых данных. Все эти документы хранятся в YT, откуда еще нужно сначала получить необходимые данные, чтобы потом обработать их.

Каждый день сотрудники Компании проверяют правильность работы счетчиков и метрик в выдаче поиска (или же используют эти данные для других задач). Они ищут в большом наборе данных необходимые документы и пытаются достать из них требуемую информацию.

Ниже приведен список недостатков такого подхода к поиску данных:

- нет единого интерфейса доступа к данным;

- каждый раз необходимо реализовывать свой обход документа, на который разработчик/аналитик тратит много времени;
- используются разные инструменты для обхода документа, и не всегда другой человек может понять чужой код, поэтому он вынужден писать свою реализацию поиска;
- написанный код может быть очень плохо оптимизирован либо не сразу работать, что приводит к потере времени на поиск ошибок в написанном скрипте;
- если же аналитик ищет необходимые данные вручную без написания скриптов и программ, то на это уходит очень много временных ресурсов;
- в ситуации, когда необходимых документов много (десятки, тысячи, миллионы, миллиарды), на поиск и последующий анализ данных потребуется очень большое количество времени;
- чем больше критериев при отборе документов, тем сложнее их получить из YT.

YT и YQL

Yandex Query Language (YQL) — универсальный декларативный язык запросов к системам хранения и обработки данных.

С помощью YQL и YT [25, 29] можно за достаточно короткое время получить все документы, которые будут относиться к конкретному поисковому ответу.

Однако, если же разработчикам будут необходимы данные о выдаче поиска, которые будут содержать в объектном ответе «колдунщик» картинок, функциональность YT и YQL не поможет справиться с данной задачей за короткое время.

«Колдунщики» поисковой системы Яндекс — это элементы поисковой выдачи, которые предлагают пользователю ответ на его запрос прямо на странице с результатами поиска.

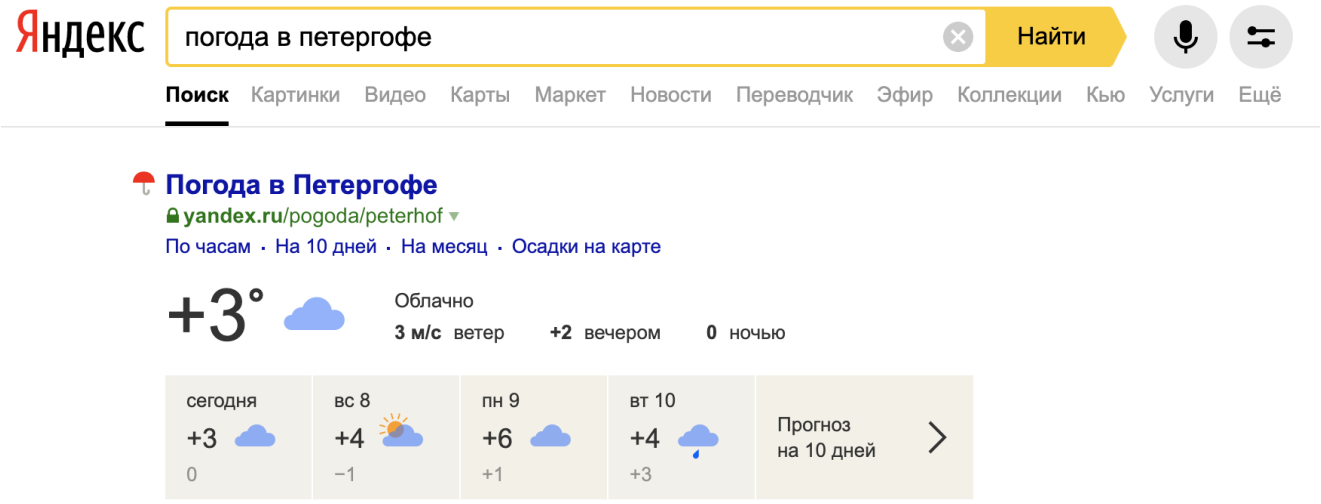

Рис.1: Пример «колдунщика» погоды в поисковой системе Яндекс.

Один из вариантов решения данной проблемы заключается в поиске совпадений по ключевым словам в документе, но это приводит к тому, что некоторые данные, которые изначально были необходимы разработчику, не попали в результат и наоборот. В итоге уже во время работы с данными разработчик понимает, что какая-то часть нужных ему данных не пришла, а какие-то данные он вообще не должен был обрабатывать.

Так как эти проблемы возникли в Компании относительно недавно, и появилась острая необходимость в едином инструменте для поиска данных, я решил взяться за эту задачу: разработать единый и удобный интерфейс для поиска и анализа необходимых документов в большом наборе данных.

Данную задачу я нашел очень интересной и актуальной, так как если разработчики получат единый и удобный интерфейс поиска данных, то работа разработчиков по проверке поисковых метрик ускорится в несколько раз.

Сейчас среднее время, уходящее на написание скриптов и запросов, у разработчиков составляет 30 минут. Стоит еще учитывать время, которое потратит разработчик на анализ данных, которые изначально не соответствовали пожеланиям разработчика/аналитика и попали в результат из-за ошибки в написанном коде. Данную оценку времени необходимо уменьшить в несколько раз.

Постановка задачи

Цель работы - разработать приложение для сотрудников компании Яндекс, которое позволит искать в заранее указанных документах необходимые данные (ссылки на счетчики и т.д.), и использовать это приложение для поиска и анализа большого количества данных на платформе YT с помощью языка запросов YQL [25, 29]. Чтобы достичь поставленной цели, необходимо решить следующие задачи:

1. Выбрать инструмент, на основе которого будет происходить поиск данных в документе - выбрать технологии, которые предоставят единый интерфейс для поиска данных (например, язык запросов).

2. После выполнения пункта 1, реализовать поиск узлов в документе - разработать программу, которая по заданным входным данным будет производить поиск в документе и возвращать результат.

3. Разработать интерфейс для отображения документов и поиска данных - предоставить сотрудникам Компании единый и удобный интерфейс, на котором будет отображены документы и все необходимые инструменты для поиска по запросам.

4. Реализовать в интерфейсе поиск по выбранному в пункте 1 решению и сделать отображение ответа для пользователей - разработать решение, которое будет отображать пользователям результаты поиска на реализованной странице с решением.

5. Использовать готовое решение из пункта 2 в поиске документов в YT - с помощью средств сборки кода добавить реализацию поиска в YQL запрос и получить результат.

Требования к задачам:

- XPath [2] парсер и поиск в документе реализовать на языке программирования TypeScript [18];
- интерфейс реализовать с помощью библиотеки React [16] и языка программирования TypeScript (TSX) [18];
- для поиска данных в YT необходимо собрать проект при помощи Webpack [18] в один скрипт и вставить в YQL [25] запрос.

Обзор литературы

Основными источниками информации при выборе и разработке языка запросов были книги Eric van der Vlist «XML Schema» [1] и John E. Simpson «XPath and XPointer» [2]. В первой книге подробно описана структура XML-документов. Во второй книге есть вся информация про язык запросов XPath:

- семантика языка;
- синтаксис;
- правила;
- возможности языка;
- примеры использования.

Во время изучения литературы [2] были выведены основные преимущества языка запросов XPath:

- это стандартизированный язык;
- XPath имеет простой синтаксис;
- синтаксис языка отличен от языка разметки XML, что позволяет использовать любой формат хранения данных (данные о счетчиках хранятся в отличном от XML формате);
- данный язык запросов очень гибкий, что позволяет, написав несложный запрос, получить из документа все необходимые данные;
- XPath полностью соответствует предметной области.

Кроме того, были изучены документации по CSS селекторам [7] и JSONPath [10]. После изучения данных источников были установлены основные недостатки данных решений и было решено использовать XPath [2] в качестве языка запросов в разрабатываемом приложении.

В ходе разработки библиотеки поиска по XPath запросу были прочитаны несколько книг по алгоритмам и структурам данных. Основным источником реализованных в работе алгоритмов являлась книга «Алгоритмы», С. Дасгупта, Х. Пападимитриу, У. Вазирани [3]. В ней хорошо описаны алгоритмы обхода древовидных структур данных, а также приведены примеры. После прочтения данной книги не составило труда реализовать алгоритм поиска данных в документе по введенному XPath запросу [2].

В ходе работы над приложением была изучена документация всех необходимых средств:

- xpath-analyzer [22] - готовая библиотека пакетного менеджера npm (Node Package Manager) для парсинга XPath запросов;
- HTML и CSS [4, 7];
- языки программирования JavaScript [5, 8, 10] и TypeScript [18];
- библиотека React [16];
- тестовые фреймворки:
    - MochaJS [15] и ChaiJS [6] - для модульных тестов;
    - HermioneJS [9] - для интеграционных тестов.
- Webpack [20] - для последующего интегрирования разработанного решения в систему YT.

После изучения всей документации, не составило труда реализовать приложение для поиска данных по XPath запросу.

Для работы с YQL были исследованы документация по YQL [25] и несколько статей [29] (основная часть статей доступна только для сотрудников компании Яндекс), с помощью которых стало понятно, как можно вставлять в YQL запрос написанный код для фильтрации данных.

Глава 1. Подготовка к решению задачи

В данной главе будет описана работа по подготовке к разработке решения, анализ имеющихся данных, выбор стека технологий.

В первом параграфе рассказывается об общем анализе имеющихся данных, об их форме. Был сделан вывод о едином строении документа и его элементов.

Во втором параграфе описывается весь стек технологий, который будет необходим для решения задачи.

В первом разделе выбирается язык запросов. Было решено использовать язык запросов для поиска данных. Рассматривались три варианта: CSS Selectors, JSONPath и XPath [1, 2, 7, 12, 23]. После рассмотрения всех вариантов и изучения литературы [1, 2], было принято решение использовать XPath в качестве языка запросов.

Второй раздел содержит информацию о реализуемом интерфейсе, который должен отображать документ и давать возможность осуществлять поиск в нем.

В третьем разделе ставится задача поиска данных в системе YT с помощью языка запроса YQL [25].

1.1 Анализ имеющихся данных

Перед решением задачи был проведен анализ имеющихся данных (документов).

При помощи YT и языка запросов YQL [25] были получены данные, которые были проанализированы. После анализа документов, было установлено, что:

- все данные имеют вид строго структурированного дерева
- у каждого узла есть поля:

- name - имя узла;
- id - уникальный идентификатор узла;
- attrs - объект с атрибутами узла (счетчики и так далее);
- children - массив дочерних узлов, каждый из которых имеет точно такую же структуру.

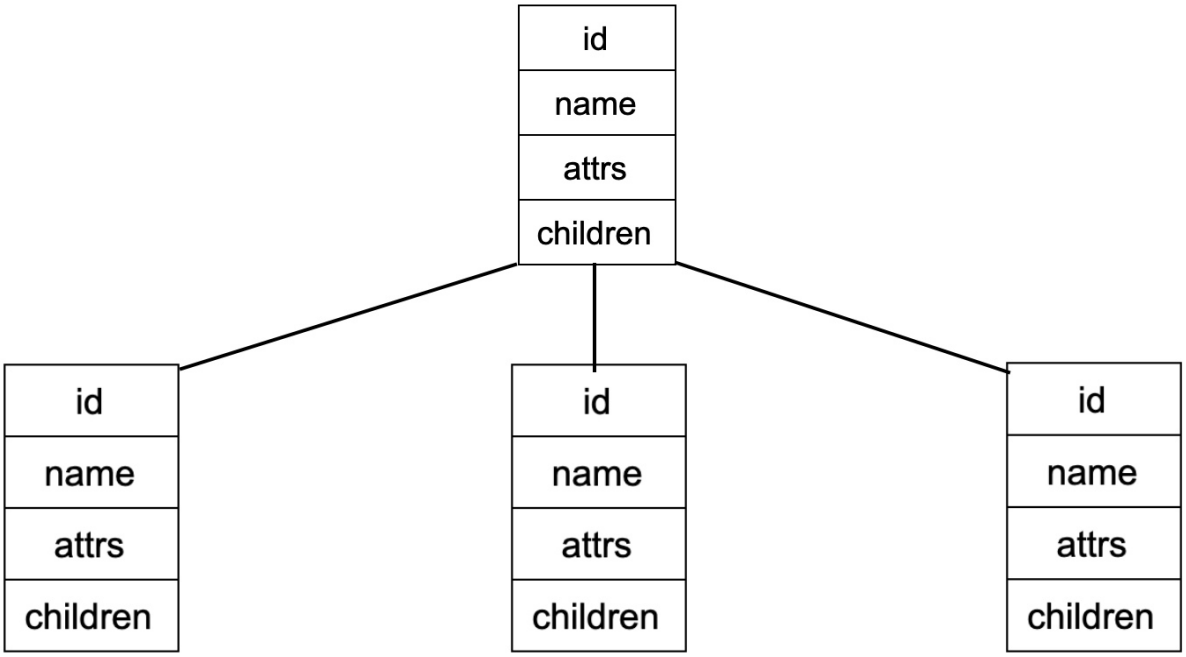

Рис. 2: Пример дерева данных.

После небольшого анализа, задача получила некоторые очертания: необходимо реализовать удобный поиск узлов в дереве по определенному условию.

Теперь необходимо выбрать стек технологий и решить, какой для этого необходим интерфейс.

1.2 Стек технологий

1.2.1 Язык запросов

Для того, чтобы производить поиск необходимых узлов в дереве, нужны входные данные. Например, в качестве входных данных может выступать строка, проанализировав которую, можно определить условия, под которые будут попадать только искомые данные в документе. В качестве строки было решено использовать запрос, по которому будет происходить поиск узлов.

Для языка запросов необходим удобный интерфейс, в котором можно будет очень быстро разобраться и начать его использовать. Нужна единая система поиска узлов. Например, пользователь вводит запрос, и программа, проанализировав этот запрос (сделав парсинг запроса), определяет, какие узлы необходимо показать (выделить).

В ходе исследования были выделены три возможных варианта языков запросов:

- CSS Selectors [7] – CSS (Cascading Style Sheets - каскадные таблицы стилей) селекторы, которые используются при создании макета сайта, могут иметь достаточно сложную структуру и вложенность;
- JSON Path [12] – аналог языка запросов XPath (XPath для JSON);
- XML Path Language (XPath) [2, 23] – язык, разработанный специально для использования с XML и применяемый для поиска узлов и наборов узлов XML-документа.

После прочтения документации и изучения литературы [1, 2] по каждому из возможных вариантов решения [7, 12, 23] были установлены следующие недостатки первых двух возможных вариантов решения.

Недостатки CSS Selectors:

- другая область применения: CSS селекторы, в первую очередь, предназначены для стилизации HTML-страниц;
- чтобы изучить синтаксис CSS селекторов разработчикам понадобится большое количество времени;
- сложный синтаксис;
- отсутствие примеров работы;
- ограничения на синтаксис (влияющие на смысл пробелы в запросах, отсутствие форматирования правил);
- сложные запросы тяжело читаются, в них легко допустить ошибку, и они не всегда понятны другим людям;
- не соответствует предметной области.

Недостатки JSON Path:

- синтаксис полегче, чем у CSS Selectors, но всё равно он не всегда прост для понимания;
- малое количество примеров работы с JSON Path;
- малое количество готовых решений для парсинга JSON Path запросов;
- не все разработчики знакомы с JSON Path.

Исходя из всего вышеописанного и пообщавшись с коллегами, было принято решение использовать язык запросов XPath для поиска данных в документах. Пользователь вводит XPath запрос и получает необходимый ему результат.

Ниже приведены причины, почему был выбран именно язык запросов XPath:

- удобный интерфейс;

- простота языка, несложный и простой в изучении синтаксис (практически все разработчики в той или иной степени знакомы с синтаксисом XPath);
- язык XPath - одновременно мощный и гибкий инструмент для разработки (большой набор функций и т.д.);
- язык стандартизирован и у него большая поддержка;
- много примеров реализации;
- скорее всего, уже есть готовые реализации для работы с XPath запросами;
- структура XPath соответствует предметной области;
- универсальность - можно сделать реализацию практически на любом языке программирования;
- не зависит от способа хранения данных (данные о счетчиках хранятся не в XML формате).

1.2.2 Интерфейс

Необходимо реализовать интерфейс отображения документа, который был бы прост в понимании и использовании сотрудниками Компании.

Было решено для страниц поиска реализовать интерфейс, на котором отображаются документ в виде дерева (со всеми ссылками, на которые можно перейти) и поле ввода XPath запроса, по которому выполняется поиск в документе.

К каждой странице поискового ответа будет прикрепляться ссылка на реализованное решение.

У данной реализации есть несколько плюсов:

- единый и удобный интерфейс;

- нет необходимости искать какой-то определенный документ: достаточно просто знать, к какому поисковому ответу принадлежит это дерево;
- по заданному запросу на этой странице отображаются искомые узлы документа.

1.2.3 YQL и YT

В YQL (Yandex Query Language) и YT [25, 29] нет возможности сделать удобный и единый интерфейс для отображения и поиска документов. Разработчикам необходимо находить в базе данных документы, которые будут удовлетворять какому-то определенному условию (например, в ответе должен быть «колдунщик» картинок).

После прочтения документации по YT и YQL [25, 29], было установлено, что в YQL есть возможность при составлении запроса добавлять функции, написанные на языках программирования Python, C++ или JavaScript, с помощью функций-обёрток, и данные будут фильтроваться через эти самые функции.

Исходя из всего вышеописанного: необходимо написать YQL запрос, добавить свои функции для обработки XPath запроса, добавить их в условие запроса и получить ответ, который будет содержать все документы, удовлетворяющие запросу.

Глава 2. Обзор существующих решений

Перед началом работы, было проведено исследование на наличие готовых решений, которые реализовывали поиск по XPath запросу в документе.

Преимущества готовых решений:

- значительная экономия времени;
- готовое решение может полностью покрывать требования задачи;
- возможность немного переделать решение под требования своей задачи;
- как правило, существующие решения оптимизированы по производительности, покрыты модульными тестами и у них есть примеры использования.

В данной главе будут продемонстрированы решения, которые удалось найти на просторах Интернета:

- встроенная реализация для работы с XPath запросами в языке программирования JavaScript [24];
- решение libxml от компании Google [13].

В конце главы сделан вывод о том, что данные решения не подходят к задаче, поэтому необходимо реализовать свою библиотеку для поиска по XPath запросу.

2.1 Встроенные возможности языка JavaScript

В языке программирования JavaScript существует встроенная реализация языка запросов XPath [2, 24].

Эта реализация языка XPath направлена на работу исключительно с XML-документами и для работы с другими форматами требуется предварительная конвертация в XML [1, 21].

На данный момент документы хранятся в JSON (JavaScript Object Notation) формате [11]. Возможно, в будущем формат хранения изменится, и поэтому необходимо, чтобы решение не зависело от формата хранения данных.

После прочтения документации [24] этой реализации и общения с наставниками проекта, были выведены основные минусы данного решения:

- это решение работает только с XML форматом [21] - на данный момент информация о счетчиках хранится в JSON формате (со временем формат хранения данных может измениться), необходимо универсальное решение, которое не будет ограничивать область применения;
- конвертация в XML занимает время (необходимо правильно указать атрибуты, ID узлов и т.д.) - сначала конвертация документа в XML формат, потом поиск по XPath запросу, потом конвертация обратно в JSON формат и вывод результатов;
- на большом объеме данных время работы сильно увеличивается - для каждого документа проделываются все шаги из предыдущего пункта; слишком ресурсозатратно.

Данная возможность языка программирования JavaScript не подходит для решения задачи, так как она накладывает ограничения на предметную область (строгий формат хранения данных).

2.2 libxml by Google

libxml - реализация поиска в XML [1, 21] документе по XPath запросу [2] от компании Google для браузеров Google Chrome.

Данное решение, как и предыдущее, рассчитано для работы исключительно с XML [21] документами, что накладывает ограничения на предметную область.

После прочтения документации и просмотра кода реализации [13] было сделано заключение, что и это решение не подходит для задачи, так как оно тоже накладывает ограничения на предметную область (необходима конвертация данных в XML формат).

Недостатки данного решения:

- сложная реализация;
- большой объем кода (не подходит для решения задачи в YT);
- все остальные пункты из параграфа 5.1.

2.3 Выводы

Данные решения были единственными, которые удалось найти. Остальные варианты были очень далеки от предметной области и были очень сложными в реализации.

Отсюда следует, что необходимо разработать собственное решение (библиотеку) для парсинга XPath [23] запросов и поиска данных в документе. Данное решение должно:

- быть простым в понимании написанного кода - для дальнейшего его улучшения и, возможно, последующего переписывания на другие языки программирования;

- быть покрыто модульными и интеграционными тестами - чтобы минимизировать риски ошибок программы и несчастных случаев во время работы (неправильные ответы или остановка программы из-за ошибок во время исполнения кода);
- практически не иметь внешних зависимостей - решение должно быть легковесным, чтобы его можно было без проблем внедрять в другие проекты, не загружая при этом много зависимостей.

Глава 3. Реализация

В данной главе подробно описаны все этапы разработки решения: XPath запросы [2], поиск в документе [3], разработка интерфейса [16. 18], работа с YQL [25].

В первом параграфе рассказывается о первом этапе реализации - чтении XPath запросов. В качестве вспомогательного инструмента была выбрана библиотека пакетного менеджера npm (Node Package Manager) - xpath-analyzer.

Во втором и третьем параграфе подробно описаны этапы разработки алгоритма поиска (идея алгоритма, поиск, проверки условий, реализация, добавление элементов в ответ) и проблемы, которые имели место быть при разработке (неограниченное количество узлов между двумя определенными, проблема индексации узлов в документе).

Четвертый параграф повествует о разработки интерфейса для решения, реализованном в первых трех пунктах. Для разработки интерфейса была использована библиотека React [16], разработанная компанией Facebook.

В пятом параграфе описано решение для YQL и YT [25] при помощи сборщика Webpack и минификатора UglifyJS [19, 20].

3.1 Чтение XPath запросов

Первая часть решения - анализ и парсинг XPath запроса [23]. В первую очередь, было решено поискать готовые решения.

Использование готового решения - хорошая альтернатива написанию кода, потому что:

- это значительно экономит время;
- готовая библиотека может полностью покрывать требования задачи;

- готовые библиотеки, как правило, оптимизированы по производительности, покрыты модульными тестами и у них есть примеры использования.

Пожелания для готового решения:

- простота реализации;
- удобный интерфейс;
- легковесность (иметь как можно меньше зависимостей).

Поиск решений производился в пакетном менеджере npm (Node Package Manager). С помощью npm решения удобно внедрять в проект.

Перебрав множество готовых библиотек для парсинга, было решено остановиться на библиотеке **xpath-analyzer** пакетного менеджера npm.

Опираясь на документацию библиотеки **xpath-analyzer** [22], были выведены недостатки других библиотек по сравнению с xpath-analyzer:

- большой объем библиотеки (много зависимостей, которые загружаются при помощи npm, что приводит к увеличению проекта и объема занимаемой им памяти);
- неудобный интерфейс отображения результата парсинга запроса (лишние поля в ответе или же неудобные для последующей обработки данные);
- неполная поддержка синтаксиса (нет поддержки стандартных функций языка запросов XPath и т.д.);
- сложная реализация парсинга.

Ниже приведены основные причины, по которым была выбрана библиотека **xpath-analyzer**:

- она занимает мало места в памяти;

- она имеет всего лишь одну зависимость от другой библиотеки, которая, в свою очередь, не имеет никаких зависимостей;
- понятная реализация парсинга;
- полная поддержка синтаксиса XPath [2];
- удобный и понятный интерфейс полученных данных после обработки запроса.

Парсинг выражений происходит на основе регулярных выражений, что довольно быстро по времени.

Библиотека предоставляет удобный интерфейс использования для разработчиков. Она не выводит лишней информации. Полученные данные после парсинга имеют структуру массива объектов, где каждый объект отвечает одному уровню запроса.

Структура этого массива:

- name - название узла (если сейчас рассматривается узел) либо специальное зарезервированное значение для случая «//» (рассматривается в параграфе 3.3);
- type - тип (узел или некоторое условие: or, and и т.д.);
- axis – имеет значение child для случая, когда проверяется вхождение какого-то узла в другой узел;
- predicates - условия (например, поиск по определенному атрибуту, поиск по индексу или же проверка на вхождение в узел какой-либо цепочки дочерних узлов);
- lhs - левая часть условия (если есть), содержит в себе все те же поля, что и текущий объект;
- rhs - правая часть условия, тоже самое, что и для lhs;

- functionName - имя функции (count, sum и т.д.), в качестве аргументов принимает lhs и rhs, если это функция от двух переменных, либо просто объект узла).

3.2 Поиск данных в дереве

Задача с парсингом XPath [2, 23] запроса решена. По массиву *steps,* полученному при помощи библиотеки пакетного менеджера npm (Node Package Manager) xpath-analyzer [22], необходимо реализовать поиск данных в документе по XPath запросу.

3.2.1 Идея алгоритма

Так как все данные документа строго структурированы и имеют вид дерева, к данной задаче очень подходит поиск в глубину от корня дерева вниз к листьям. Спускаясь от корня вниз к листьям, необходимо сравнивать имя узла в документе с именем узла в запросе. Поиск в глубину поможет сохранить порядок отображения найденных узлов «сверху-вниз», что будет очень удобно для пользователей приложения.

Также, если имеются условия, необходимо сравнивать атрибуты узла в документе с атрибутами, прописанными в запросе. Если на каком-то из этапов спуска становится понятно, что данный узел и его дочерние элементы не подходят под запрос (не совпадают имена или не выполняются условия), то поиск в данном узле прекращается и переходит к остальным в порядке рекурсии. Результатом работы алгоритма будет массив узлов, отсортированный «сверху-вниз» для удобного отображения пользователям.

3.2.2 Реализация

Для поиска необходимых узлов в дереве используется поиск в глубину DFS (Depth-First Search) [3].

Это гарантирует, что в результирующем списке узлы будут удобно отсортированы. Это необходимо, чтобы осуществить навигацию по найденным узлам на странице с документом «сверху вниз», как это реализовано в других известных поисках на странице (например, поиск слова на какой-либо странице в браузере идет по документу «сверху вниз»).

3.2.2.1 Точка входа

Поиск начинается с корня дерева. На первом этапе алгоритм решения получает на вход поле «steps[0].type». Возможные варианты:

- данное поле является условием; тогда входная функция вызывается рекурсивно для левой и правой части условия (для полей «steps[0].lhs» и «steps[0].rhs» соответственно), условия (OR, AND) всегда делятся на левую и правую части, поэтому с ними удобно работать рекурсивно;
- указанное поле - функция; в данной ситуации необходимо вызвать соответствующий метод для этой функции (который в свою очередь потом вызовет для корня дерева этот же метод);
- узел; осуществлять рекурсивный поиск вглубь дерева с проверкой условий.

3.2.2.2 Проверка условий

На каждом этапе поиска первым делом проверяется совпадение полей «name» у узла и у значения в запросе. Если имена не совпадают, то поиск в

данном узле прекращается, и продолжается в других узлах в порядке рекурсии. После проверки поля «name» проверяются условия (если они есть):

- если это проверка атрибутов, то ведется поиск по атрибутам узла на совпадение условия;
- если это проверка на вхождение дочерних элементов, то среди детей текущего узла ищется узел с нужным именем.

Если поля «name» совпадают и все условия выполняются, поиск продолжается для всех дочерних узлов текущего узла.

3.2.2.3 Добавление в ответ

Если текущий узел удовлетворяет XPath запросу, то в результирующий массив добавляется значение в поле «id» текущего узла. В результате удобно хранить идентификаторы, поскольку такая структура данных будет занимать немного места в памяти и к узлам в документе будет удобно обращаться через уникальный идентификатор.

Результат работы алгоритма: отсортированный «сверху-вниз» массив идентификаторов узлов, удовлетворяющих запросу.

3.3 Проблемы, появившиеся при реализации поиска

3.3.1 Неограниченное количество узлов между двумя определенными

На данный момент одна из главных проблем реализации – это ситуации, когда в запросах встречается запись «//». Она означает, что на месте пропуска может быть неограниченное (≥0) число узлов. Такая ситуация вызывает неопределенность для ранее написанного решения. Необходимо рассмотреть все возможные случаи и добавить в исходное решение обработку подобных случаев.

Например, дан запрос «node1//node3». Неизвестно, какое количество узлов существует между указанными двумя, поэтому необходимо перебрать все варианты:

1. Между двумя данными узлами узла нет: «node1/node2».
2. Узел ровно один: «node1/node3/node2».
3. Узлов много (>1): «node1/node3/node4/node5/node2».

Такой случай порождает многократные рекурсивные вызовы, так как на каждом следующем шаге необходимо рассматривать этот случай.

В данной ситуации библиотека **xpath-analyzer** [22] в поле «name» вставляет специальное значение «NODE».

«NODE» - специальное зарезервированное библиотекой имя [22], которое показывает, что в запросе вместо имени узла стоит строка вида «//» (в документе нет узлов с таким именем, поэтому зарезервированное имя не накладывает ограничения на структуру документа).

В ситуациях, когда узел отсутствует, происходит 3 рекурсивных вызова, которые в свою очередь для дочерних узлов вызывают по 3 рекурсивных вызова. Для всех подходящих запросу узлов, значение их поля «id» записывается в результирующий массив «resultTreeList».

Из-за таких случаев время работы алгоритма увеличивается. Если раньше необходимо было делать обход дерева один раз, то теперь для узла и всех его детей нужно делать по 3 таких обхода.

3.3.2 Индексация узлов в дереве

Еще одна проблема - индексация узлов в дереве. Например, дан запрос: «/node1/node2[2]»

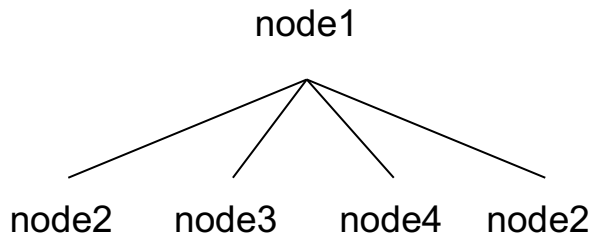

Рис. 3: Структура дерева.

Данная ситуация возникает, когда в документ, построенный по объектному ответу, добавляются узлы, которые были подгружены на страницу не сразу, а через какое-то время или же после какой-то активности на странице от пользователя.

Необходимо правильно реализовать индексацию узлов, чтобы к ним можно было обращаться в запросе через индексы.

Так как порядок узлов в дереве произвольный (это связано с тем, что некоторые узлы в документ записываются не сразу, а через какое-то время, например, когда страница с ответом полностью загрузилась), было решено использовать дополнительную хэш-таблицу [3], ключом в которой будет абсолютный путь до узла, а значением - список узлов, у которых этот абсолютный путь одинаковый. Такой список узлов будет правильно отсортирован, что гарантирует правильный доступ через индекс. Теперь к узлам можно обращаться в запросе через индексы.

С помощью этой хэш-таблицы [3] можно сохранять данные для последующего переиспользования: не будет нужды заново обходить документ в поисках узлов. Если запрос достаточно прост (без условий и будет совпадать с одним из ключей в хэш-таблице), то достаточно просто по заданному XPath запросу найти все узлы в хэш-таблице.

3.4 Реализация интерфейса

Разработка интерфейса происходила на языке программирования TypeScript и TSX (TypeScript XML) [18] при помощи библиотеки React [16], разработанной компанией Facebook. Данная библиотека легковесная и простая в использовании, и с ее помощью можно быстро разработать одностраничное решение.

TypeScript XML (TSX) — это расширение синтаксиса TypeScript, которое позволяет использовать похожий на HTML (HyperText Markup Language) [4] синтаксис для описания структуры интерфейса (источник - Википедия).

Моменты, на которые необходимо было обратить внимание при разработке страницы с решением:

- необходимо было решить, как правильно отправлять JSON (JavaScript Object Notation) [11] документ на страницу с решением поиска по XPath [2, 23] запросу для последующего отображения и поиска;
- нужно решить, как лучше и правильнее всего отображать подходящие запросу узлы в документе и разработать это решение;
- наилучший способ хранения узлов на странице с приложением для быстрого к ним доступа.

3.4.1 Отправка JSON дерева на страницу с решением поиска по XPath

Так как интерфейс с JSON документом открывается на другой странице, необходимо каким-то образом осуществить отправку документа на страницу. Было решено при нажатии на ссылку с переходом отправлять данные с помощью встроенных возможностей языка программирования JavaScript:

метод window.postMessage() [4, 5, 8, 10]. В данную функцию передается адрес (url) страницы с решением и сам JSON (JavaScript Object Notation) документ.

На самой же странице с документом это сообщение отлавливается с помощью метода window.addEventListener(«message», callback). В функцию callback приходит JSON [11] документ и отображается на странице для последующего в нем поиска данных по XPath запросу.

3.4.2 Отображение подходящих запросу узлов

Изначально отображаются только 1-й и 2-й уровни дерева. При открытии страницы с документом отображается корень дерева и его дочерние элементы первого уровня. При клике на элемент или же навигации к нему после поиска, элемент «раскрывается», т.е. показывает свои дочерние элементы первого уровня. Во время навигации по узлам после поиска по запросу, текущий узел подсвечивается красной рамкой и «раскрывается» на один уровень.

Для быстрого переключения между найденными узлами, были добавлены кнопки навигации. Для массива идентификаторов узлов хранится указатель на текущий (активный) id, и при нажатии [4] на кнопки навигации, указатель сдвигается влево или вправо. Это позволяет быстро получать и смотреть результаты, а также не использовать лишние ресурсы.

3.4.3 Хранение узлов и быстрый доступ к ним

Еще одна проблема: хранение информации об узлах на странице с приложением и быстрый доступ к найденным узлам на странице с документом. Необходимо реализовать это так, чтобы поиск нужного узла не

занимал много времени, и не выделялось слишком много памяти на поиск и хранение информации об узлах.

Разработанное решение: при отображении дерева к каждому узлу добавлять атрибут «id» - уникальный идентификатор (поскольку он обязательно есть у каждого узла).

Каждый узел в DOM-дереве (Document Object Model) [4] будет иметь уникальный идентификатор (один узел документа соответствует одному узлу в дереве данных), который необходимо будет найти.

Когда же будет необходимость найти определенный узел, используется встроенный метод языка JavaScript для работы с DOM деревом document.getElementById(*id*) [4].

Данный метод возвращает ссылку на объект типа Element соответствующий указанному ID или null, если элемент с указанным ID не найден в документе.

Оценка сложности метода получения HTML (HyperText Markup Language — «язык гипертекстовой разметки») элемента document.getElementById(*id*): *O(1).*

Следовательно, операция «подсветки» необходимого узла будет выполнена за константное время. Данное решение оптимально по требованиям и по производительности.

Так, имея HTML [4] элемент узла остается только выделить его и показать все дочерние элементы, добавив несколько свойств с помощью CSS стилей, добавив ему CSS (Cascading Style Sheets) селектор при помощи методов языка программирования JavaScript [5].

3.5 Поиск в YQL

В YQL (Yandex Query Language) есть возможность добавления в запросы своих функций для работы с данными [25], написанных на языках программирования C++, Python или JavaScript. Для этого необходимо написать функции-обёртки, которые будут связывать код запроса с кодом, который был написан в предыдущих параграфах. Далее эти функции-обёртки вставляются в YQL запрос [25].

Необходимо написанный в прошлых параграфах код собрать в один большой файл (скрипт) и добавить в YQL [25] запрос. Осуществить это можно с помощью сборщика Webpack [20].

Webpack - статический модульный сборщик для приложений на JavaScript.

С помощью Webpack [20] и его плагинов был собран весь написанный код, сжат и оптимизирован, тем самым был уменьшен его размер.

Это необходимо, чтобы вставляемый код не занимал много места и быстрее бы отправился через сеть.

В качестве минификатора был использован плагин для Webpack - UglifyjsWebpackPlugin [19]. Минификатор необходим для уменьшения объема занимаемой памяти кода.

Теперь необходимо написать в YQL [25] функции, которые будут добавлены в запрос.

```
$list = Javascript::generate(Callable<()->List<Utf8>>, $script);
$xpathFromList = Javascript::xpathFromList(Callable<(Utf8?,List<Utf8>)->Utf8?>, $script);
```

Рис. 4: Функции обёртки.

```
SELECT request FROM 'Название хранилища'
WHERE json_blocks != '{}' and $xpathFromList(CAST(json_blocks AS Utf8), $list()) != ''
```

Рис. 5: YQL запрос.

1. $list() - функция, возвращающая список XPath запросов [2].
2. $xpathFromList(block: string, list: string[]) - функция, проверяющая, подходит ли данный документ под один из запросов.

Выводы

Что уже сделано?

В рамках данной работы была поставлена цель: разработать приложение для сотрудников компании Яндекс, которое будет позволять находить в заранее указанных документах необходимые данные (ссылки на счетчики и т.д.), а также использовать это решение для поиска и анализа большого количества данных. В ходе работы были получены следующие результаты:

1. Рассмотрены возможные варианты языков запросов: XPath, JSON Path и CSS Selectors [2, 7, 12]. После изучения каждого варианта были выведены основные преимущества и недостатки данных решений. В качестве языка запросов был выбран XPath [2], поскольку данный язык запросов лучше остальных двух подходил к предметной области.

2. Сделан обзор существующих решений задачи поиска данных в документе по XPath (XML Path Language) запросу [1, 2, 23]. В ходе обзора существующих решений было установлено, что ни встроенные возможности языка JavaScript [24] для работы с XPath запросами, ни библиотека libxml [13] от компании Google не подходят для решения задачи, так как они ограничивают предметную область и требуют строго указанного формата хранения данных.

3. Исследован пакетный менеджер npm (Node Package Manager) на предмет наличия в нем готовых парсеров XPath запросов. В npm была найдена готовая библиотека для парсинга XPath запросов **xpath-analyzer** [22]. Библиотека полностью удовлетворяет требованиям задачи, покрывает предметную область и предоставляет удобный интерфейс.

4. Разработано решение, которое находит данные в документе по XPath запросу. Алгоритм показал отличные (<10ms) результаты на основе имеющихся данных.

5. Реализован интерфейс для отображения документа и поиска в нем данных по XPath запросу. При помощи библиотеки React [16], разработанной компанией Facebook, был успешно разработан и внедрен интерфейс отображения документов.

6. Разработан инструментарий на языке JavaScript для поиска больших данных в YT и YQL (Yandex Query Language) [25]. С помощью сборщика Webpack [20] и минификатора UglifyJS [19], написанный код был собран и добавлен в YQL запрос [25], были написаны функции обёртки для JavaScript кода.

Суммируя вышеупомянутые результаты, можно сделать вывод, что поиск готовых решений и алгоритмов в данной задаче оправдан и что с помощью готовых инструментов можно разработать полноценный продукт, который будет полностью покрывать требования поставленной задачи.

Интерфейс для отображения

MVP (minimum viable product - минимально жизнеспособный продукт) успешно продемонстрирован сотрудникам Компании.

Реализация и внедрение библиотеки прошли успешно. Решение было покрыто модульными тестами. Библиотека выложена в открытый доступ. Ответ выдается мгновенно (<10ms).

Исходный код библиотеки (часть кода пришлось изменить, чтобы не нарушать соглашение о неразглашении данных) на GitHub: https://github.com/avdeev1/xpath-tree-json

Интерфейс для отображения документов реализован и внедрен. Решение было покрыто модульными, интеграционными и e2e (end to end) тестами. Модульные тесты были написаны с помощью библиотек Mocha, Chai и Sinon.js [6, 15]. Интеграционные и e2e тесты были написаны с помощью Hermione.js, которая в свою очередь использует WebdriverIO и Mocha [9].

Данное решение успешно внедрено и активно используется сотрудниками компании. Поскольку XPath [2, 23] достаточно прост в освоении, разработчики и аналитики практически сразу стали пользоваться данным решением, не потратив при этом много времени на изучение языка запросов XPath.

Работа с несколькими документами одновременно

Во время разработки интерфейса, когда библиотека уже была разработана и был почти написан код для интерфейса, появилась еще одна задача: добавить на страницу возможность работы с несколькими документами одновременно. Это добавление сделало бы анализ немного быстрее, так как можно сразу осуществлять поиск не в одном документе, а в нескольких.

После реализации последовательного обхода каждого дерева были замечены задержки во времени от начала поиска до конечного вывода результатов. Тогда было решено, что необходимо реализовать параллельные (асинхронные) обходы документов.

С помощью концепции Promise [5, 8], реализованной в языке программирования JavaScript вместо того, чтобы выполнять *N*

последовательных поисков нужных узлов в дереве, это делается параллельно (асинхронно). После выполнения Promise.all(list) на выходе будет получен массив, в котором каждый элемент - массив результатов.

Данные результаты остается только объединить в один массив уникальных идентификаторов. Это позволяет не тратить слишком много времени на поиск узлов в нескольких деревьях.

Поиск в YQL

Написана реализация для YQL [25]: с помощью сборщика Webpack [20] и минификатора UglifyJS [19] код собран в единый скрипт и вставлен с помощью функций-обёрток в YQL (Yandex Query Language) запрос.

Принцип работы YT:

1. Все данные хранятся частями в разных Дата-центрах.
2. Сервер узнает, где именно находятся эти самые части и обращается к нужным Дата-центрам.
3. Дата-центры отправляют данные на сервер.
4. Создается большое количество (десятки тысяч) виртуальных машин, на которых уже каждое дерево проходит через написанное решение.
5. Все подходящие данные собираются в один большой ответ и приходят на клиент.
6. Данные готовы к работе

Общее количество данных: ~ 10 млрд

Общее время работы:

- в рабочее время, когда сервера загружены: 19 мин.
- в выходные или же ночью, когда нагрузки на сервер меньше: 12 мин.

7. Уменьшено среднее время написания запросов для поиска:
   - до написания приложения у разработчиков уходило на написание скриптов в среднем по 30 минут, при этом это не гарантировало, что все полученные данные - те, которые изначально хотел разработчик получить;
   - после написание приложения - не больше 5 минут в среднем, т. к. основной код написан, и разработчику необходимо всего лишь указать условия, по которым необходимо фильтровать данные.

Дальнейшие улучшения приложения

Оптимизация запросов

XPath запросы [2] могут быть сколь угодно сложными. Например, для запроса *«//node[name1] or //node[name2]»* придется обходить дерево дважды, поскольку сначала происходит поиск для запроса *«//node[name1]»,* а уже потом и для *«//node[name2]».* Появляется проблема большого количества обходов дерева при множественных условиях.

Необходимо оптимизировать запросы, дабы избежать повторных обходов документа. Это улучшит быстродействие алгоритма и поможет в будущем увеличении данных. Например, стоит обращать внимание на «похожие» запросы и соединять их в один.

Например, вышеупомянутый запрос:

«//node[name1] or //node[name2]»

должен быть преобразован к запросу:

«//node[name1 or name2]»

Как один из возможных вариантов решения данной проблемы, можно смотреть на путь до какого-то условия, и если он одинаковый, то объединять *N* запросов в один.

Необходимо рассмотреть данную задачу и найти виды запросов, которые можно будет оптимизировать, чтобы избежать повторных обходов документов.

Оптимизация алгоритма

Необходимо изучить текущее решение на наличие улучшений быстродействия алгоритма поиска путём избавления от рекурсии, избавления от слишком большого количества рекурсивных вызовов. При достаточно большой глубине рекурсии браузер может просто не выдержать и «зависнуть». Поэтому необходимо из реализации убрать рекурсию и использовать стек.

Рассмотреть первую задачу: можно ли вместо объединения запросов делить запрос на два разных и выполнять поиск параллельно (асинхронно). Необходимо понять, в каких случаях выгоднее объединять запросы, а в каких делить. Сравнить время выполнения, выбрать более быстрое решение. Возможно, найти третье решение, которое будет быстрее первых двух.

Проблема множественных XPath запросов

Еще одна проблема: дано несколько XPath запросов [2, 23] и большое количество документов с данными. Необходимо оптимальным образом обработать эти документы, возможно, сохраняя какие-то промежуточные данные, чтобы избежать повторных обходов.

Заключение

В ходе исследования были подробно изучены языки запросов XPath, JSONPath и CSS Selectors [1, 2, 7, 12, 23], библиотека React [16] и язык запросов YQL (Yandex Query Language) [25], разработана библиотека для поиска данных по запросу в документе, разработан интерфейс для отображения документов. Результатами работы являются полученные выводы о сходимости и эффективности этих методов, а также инструментарий для работы с ними.

Разработанное приложение существенно уменьшило затрачиваемое разработчиками время на поиск и анализ большого количества данных счетчиков и метрик поисковых запросов. Если раньше разработчики тратили в среднем по 30 минут на написание запросов и скриптов, то после внедрения разработанного приложения среднее время работы по созданию запросов уменьшилось в 6 раз.

Разработанное приложение было переписано на другие языки программирования (Python, C++) другими разработчиками, успешно внедрено в проекты и теперь активно используется сотрудниками компании Яндекс.

В дальнейшем планируется продолжить работу по увеличению скорости быстродействия реализованного решения и рассмотрению задач, описанных в предыдущем разделе.

Исходный код на GitHub: https://github.com/avdeev1/xpath-tree-json

Список литературы

1. Литература

[1] Eric van der Vlist, «XML Schema», Publisher O'Reilly, 2002 год, 400 страниц - подробное описание строения XML документов.

[2] John E. Simpson, «XPath and XPointer», Publisher O'Reilly, 2002 год, 224 страницы - данная книга подробно описывает язык запросов XPath.

[3] С. Дасгупта, Х. Пападимитриу, У. Вазирани (Перевод с английского А. С. Куликова под редакцией А. Шеня), «Алгоритмы», Издательство МЦНМО, 2014 год, 319 страниц - работа с древовидными структурами данных и алгоритмы их обхода.

[4] И. Кантор, «Документ, события, интерфейсы», 2019 год, 309 страниц - все о DOM (Document Object Model) дереве, работа с ним, события на странице и их обработка и тд.

[5] И. Кантор, «Язык JavaScript». 2019 год, 1113 страниц - язык JavaScript, его синтаксис, спецификации, применение.

2. Ссылки на документации

[6] Chaijs documentation - документация к тестовому фреймворку chai, необходим для написания unit тестов.
https://www.chaijs.com/

[7] CSS documentation - документация по CSS селекторам, CSS селекторы были одним из вариантов языка запросов для данной задачи, было решено его не использовать из-за сложной структуры запросов.
https://www.w3schools.com/cssref/css_selectors.asp

[8] ECMAScript® 2019 Language Specification - спецификация языка программирования JavaScript, необходима для разработки библиотеки, интерфейса и написания тестов.
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf

[9] Hermionejs documentation - фрейморк для интеграционного тестирования, разработанный компанией Яндекс, был использован для покрытия тестами интерфейса отображения документов.
https://github.com/gemini-testing/hermione

[10] JavaScript book - ресурс для изучения языка программирования JavaScript.
https://javascript.info/

[11] JSON (JavaScript Object Notation) documentation - документация по JSON, была необходима, т.к. все документы на текущий момент хранятся в JSON формате.
https://www.json.org/json-en.html

[12] JSONPath documentation - документация по JSONPath, который рассматривался как язык запросов для текущей задачи. Было решено не использовать его, т.к. JSONPath имеет сложный синтаксис.
https://github.com/json-path/JsonPath

[13] libxml by Google - разработанная компанией Google библиотека для работы с XPath запросами. Было решено от нее отказаться в пользу своей реализации, т.к. библиотека ограничивает предметную область и требует конкретного формата хранения данных.
https://github.com/chromium/chromium/tree/master/third_party/libxml

[14] MDN Web docs - ресурс для разработчиков по JavaScript, где были найдены встроенные возможности в языке для работы с XPath запросами.
https://developer.mozilla.org/en-US/#

[15] Mochajs documentation - документация по тестовому фреймворку mocha, необходим для написания unit тестов, был использован для тестирования библиотеки, алгоритма поиска и интерфейса.
https://mochajs.org/api/mocha

[16] React documentation - документация по библиотеке React, которая разработана компанией Facebook. Была использована при разработке интерфейса отображения документов.
https://reactjs.org/docs/getting-started.html

[17] Selenium documentation - документация по Seleniun для написания интеграционных тестов.
https://selenium.dev/documentation/en/

[18] TypeScript specification - спецификация языка программирования TypeScript, который был необходим для написания библиотеки, алгоритма поиска и разработки интерфейса.
http://www.openwebfoundation.org/legal/the-owf-1-0-agreements/owfa-1-0

[19] UglifyjsWebpackPlugin documentation - плагин для сборщика кода Webpack, необходим для минификации написанного кода в целях уменьшения его объема.
https://webpack.js.org/plugins/uglifyjs-webpack-plugin/

[20] Webpack documentation - документация по сборщику кода Webpack. Необходим для решения задачи с поиском данных в системе YT при помощи YQL.
https://webpack.js.org/concepts/

[21] XML documentation - документация по XML (Extensible Markup Language)
https://www.w3.org/XML/

[22] xpath-analyzer documentation - документация по библиотеке npm xpath-analyzer. Библиотека была использована для парсинга XPath запроса.
https://www.npmjs.com/package/xpath-analyzer

[23] XPath documentation - язык XPath (XML Path Language), был выбран языком запросов в данной задаче, т.к. по сравнению с другими (CSS Selectors и JSONPath) прост в освоении и соответствует предметной области
https://www.w3schools.com/xml/xpath_intro.asp

[24] XPath in JavaScript - встроенные возможности языка программирования JavaScript для работы с XPath запросами. Было решено отказаться от этого решения, т.к. оно накладывало ограничения на предметную область и требовало определенного формата хранения данных.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_using_XPath_in_JavaScript

[25] Документация к YQL - изучение языка запросов YQL (Yandex Query Language). Необходим для поиска данных в YT. https://cloud.yandex.ru/docs/ydb/yql/reference/overview

3. Ссылки на сайты с полезной информацией

[26] Difference Between JSON and XML: https://www.differencebetween.com/wp-content/uploads/2017/12/Difference-Between-JSON-and-XML.pdf

[27] Tutorials point: JSON: https://www.tutorialspoint.com/json/json_tutorial.pdf

[28] Tutorials point: XPath for XML: https://www.tutorialspoint.com/xpath/xpath_tutorial.pdf

[29] YT: зачем Яндексу своя MapReduce-система и как она устроена (2016 год): https://habr.com/ru/company/yandex/blog/311104/

[30] Yet Another Conference: YT — новая платформа распределённых вычислений: https://events.yandex.ru/events/yac/2013

[31] Как писать меньше кода для MR, или Зачем миру еще один язык запросов? История Yandex Query Language (2016 год): https://habr.com/ru/company/yandex/blog/312430/

Приложение

Приложение А. Список терминов, употребляемых в работе

1. **YT** — основная платформа для хранения и обработки больших объемов данных в компании Яндекс [27]. Это MapReduce система, направленная на широкий спектр вычислительных задач в области Big Data.
2. **MapReduce** — модель распределенных вычислений, представленная компанией Google, используемая для параллельных вычислений над очень большими, вплоть до нескольких петабайт, наборами данных в компьютерных кластерах (источник - Википедия).
3. **YQL (**Yandex Query Language) [25] — универсальный декларативный язык запросов к системам хранения и обработки данных. Необходим для работы в YT.
4. **Колдунщики** поисковой системы Яндекс — это элементы поисковой выдачи, которые предлагают пользователю ответ на его запрос прямо на странице с результатами поиска.
5. **XPath** (XML Path Language) [2, 23] — язык, разработанный специально для использования с XML и применяемый для поиска узлов и наборов узлов XML-документа.
6. **JSON Path** [12] — аналог языка запросов XPath (XPath для JSON).
7. **CSS Selectors** [7] — CSS (Cascading Style Sheets - каскадные таблицы стилей) селекторы, которые используются при создании макета сайта, могут иметь достаточно сложную структуру и вложенность.
8. **HTML** (HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык разметки документов (источник - Википедия).
9. **XML** (*e**X**tensible **M**arkup **L**anguage*) [1] — расширяемый язык разметки. Может использоваться как формат хранения данных.
10. **JSON** (JavaScript Object Notation) — способ хранения данных.

11. **npm** (Node Package Manager) — менеджер пакетов, входящий в состав Node.js. Необходим для установки библиотеки **xpath-analyzer**, выбранной для парсинга XPath запросов.
12. **xpath-analyzer** [22] — библиотека пакетного менеджера npm для парсинга XPath запросов.
13. **DFS** (Depth-First Search) [3] — алгоритм обхода графа «в глубину». Необходим для реализации поиска узлов по XPath запросу.
14. **TSX** (TypeScript XML) [18] — это расширение синтаксиса TypeScript, которое позволяет использовать похожий на HTML [4] синтаксис для описания структуры интерфейса (источник - Википедия).
15. **DOM** (Document Object Model) [4] — объектная модель документа, каждый HTML-тег является объектом.
16. **Webpack** [20] — это статический модульный сборщик для приложений на JavaScript.
17. **UglifyjsWebpackPlugin** [19] — минификатор, плагин для Webpack. Необходим для уменьшения объема занимаемой памяти кода.
18. **Функции обертки в YQL** [25] — функции, которые связывают код, написанный на языке программирования JavaScript, с запросом, написанным на языке запросов YQL.
19. **MVP** (Minimum Viable Product) — минимально жизнеспособный продукт.

Приложение В. Примеры кода

В.1. Пример: работа библиотеки xpath-analyzer

Исходный запрос: «node1[node2 or node3]»

```
steps: [
 {
 "axis": "child",
 "type": "node",
 "name": "node1",
 "predicates": [{
  "type": "or",
  "lhs": {
   "type": "relative",
   "steps": [{
    "axis": "child",
    "type": "node",
    "name": "node2",
    "predicates": []
   }]
      },
  "rhs": {
   "type": "relative",
   "steps": [{
    "axis": "child",
    "type": "node",
    "name": "node3"
   }],
  "predicates": []
  }
 }
]
```

В.2. Код функции поиска в документе узлов, подходящих запросу

Поиск узлов в документе, реализованный на языке программирования JavaScript:

```
function find(steps, tree, resultTreeList) {
  const step = steps.pop();
  const predicates = step.predicates;

 if (tree.name === step.name && getPredicate(predicates, tree)) {
    if (steps.length === 0) {
      resultTreeList.push(tree.id);
    }
    tree.children.forEach(child => {
      find(steps, child, resultTreeList);
    });
  }
}
```

В.3. Код функции поиска, рассматривающий случай неограниченного количества узлов между двумя данными

```
function find(steps, tree, resultTreeList) {
  const step = steps.pop();
  const predicates = step.predicates;
  if (tree.name == step.name && getPredicate(predicates, tree)){
    if (steps.length === 0) {
      resultTreeList.push(tree);
```

```
    }
    tree.children.forEach(child => {
      find(steps, child, resultTreeList);
    });
  } else if (name === NODE && tree.children) {
    find(steps, tree, resultTreeList);
    tree.children.forEach(child => {
      if (steps.length) {
        find(steps, tree, resultTreeList);
        const stepsForChildren = steps.concat(step);
        find(stepsForChildren, child, resultTreeList);
      }
    });
  }
}
```

В.4. Код функции, сохраняющий данные по узлам в хэш-таблицы для повторного поиска

```
function checkNodesInHashTable(xpathQuery) {
  if (nodesHashTable.get(xpathQuery)) {
  return nodesHashTable.get(xpathQuery);
  }
  return false;
}
```

Приложение С. Примеры рабочего приложения

С.1. Начальный интерфейс при загрузке страницы

Рис. 6: Начальное отображение дерева перед поиском.

С.2. Поиск в документе

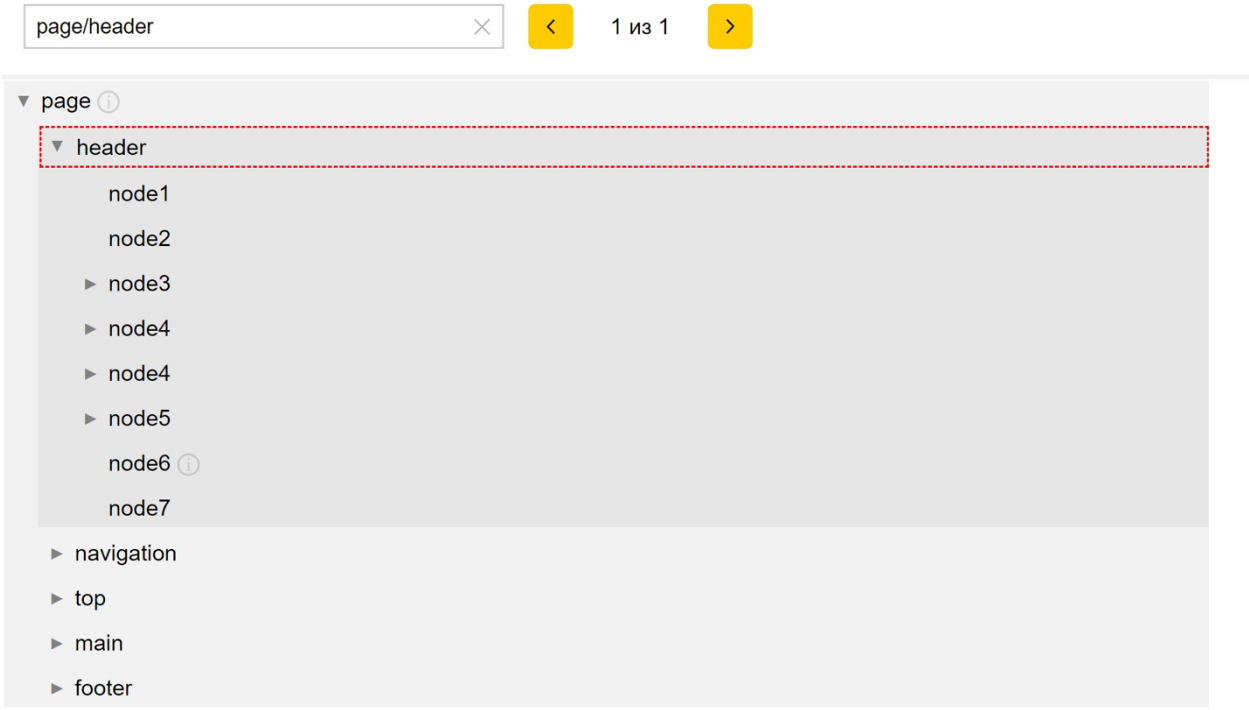

Рис. 7: Пример отображения узла после поиска по XPath запросу.

С.3. Отображение нескольких документов на странице

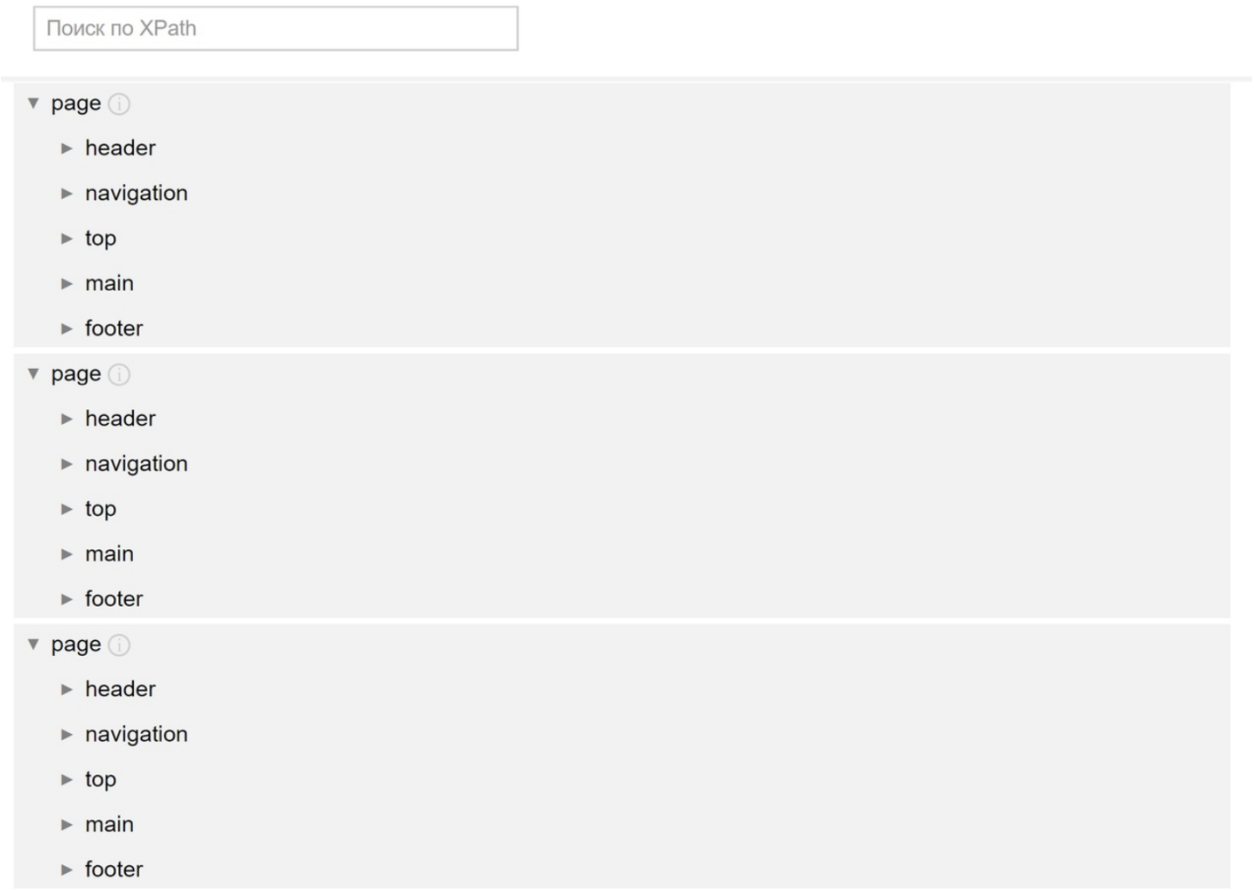

Рис. 8: Пример нескольких документов на странице.

С.4. Поиск в нескольких документах сразу

При помощи кнопок навигации можно переходить на другое дерево и смотреть на его результаты.

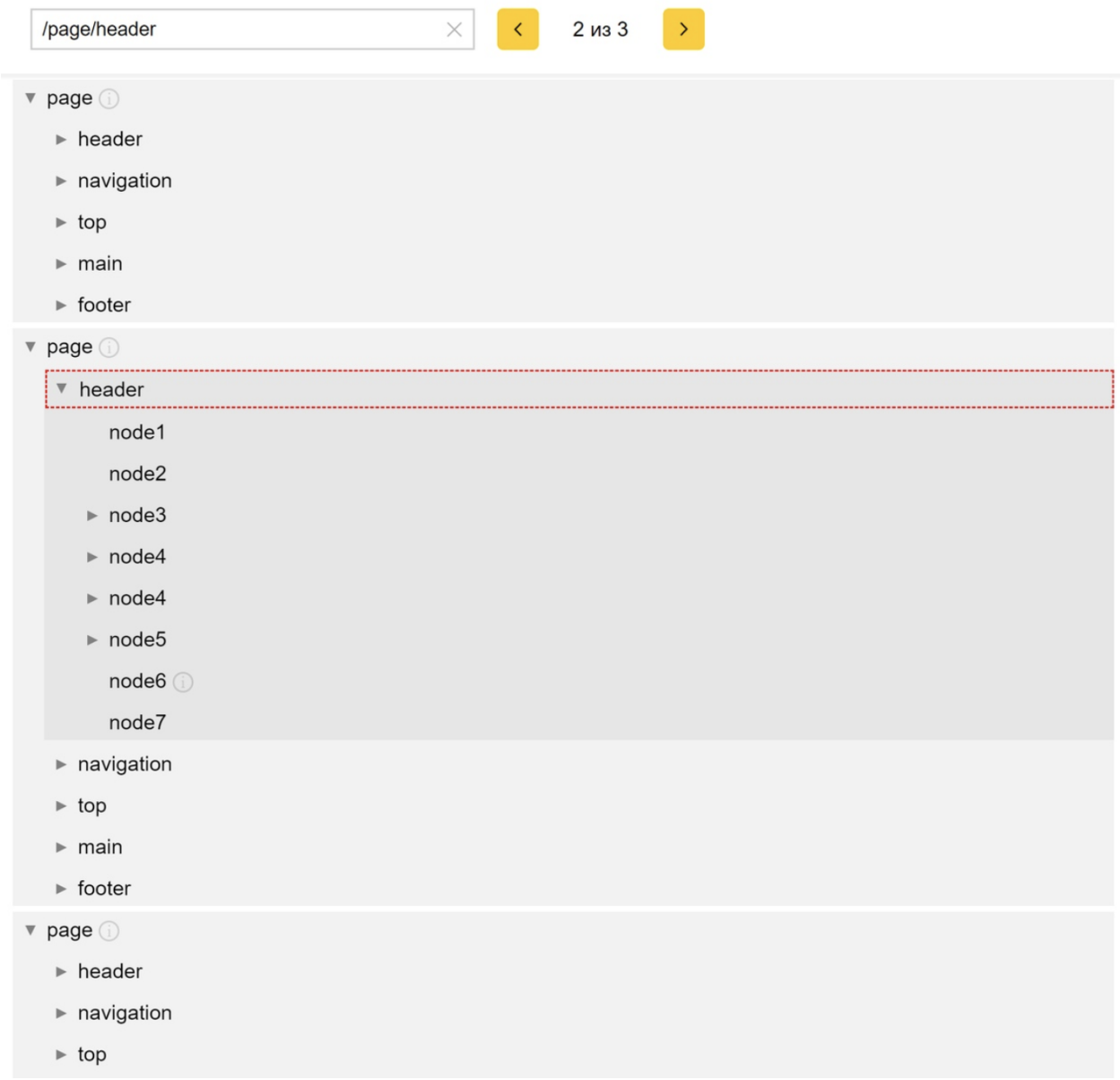

Рис. 9: Пример поиска узлов в нескольких деревьях одновременно.